

**School of Built Environment, Engineering and Computing**



**Week 4: DAV Time Series**

**By**

**Prof. Ah-Lian Kor**

**Time Series Exercises**

Time series is a series of observations recorded at a regular time interval.

1. **Introduction**

Dates and Time in python.

We shall use pandas which is superb for financial modelling. We will need to work with dates, times, and time-indexed data. Some basic concepts:

* Time stamps for particular points in the time line (e.g., July 4th, 2015 at 7:00am)
* Time interval or period consists of uniform length and does not overlap (e.g. year 1975)
* Time deltas or duration is the exact length of time which is equal to the difference between the end and start points (e.g. a duration of a second, minute, hour, day, week, month, year, decade, millennium)

**Functions**

strftime() and strptime() Behavior

date, datetime, and time objects all support a strftime(format) method, to create a string representing the time under the control of an explicit format string. Broadly speaking, d.strftime(fmt) acts like the time module’s time.strftime(fmt, d.timetuple()) although not all objects support a timetuple() method.

Conversely, the datetime.strptime() class method creates a datetime object from a string representing a date and time and a corresponding format string. datetime.strptime(date\_string, format) is equivalent to datetime(\*(time.strptime(date\_string, format)[0:6])).

**Resources**

<https://docs.python.org/3.3/library/datetime.html#strftime-and-strptime-behavior>

* 1. **Current Day, Date, and Time**

|  |  |
| --- | --- |
| Representation | Meaning |
| strftime | Returns a string representing the date.  Complete list <https://strftime.org/> |
| %A | Weekday e.g. Monday |
| %Y | Year (e.g. 2013) |
| %m | Month (e.g. 09) |
| %w | Weekday as a decimal number; 0 is Sunday…. 6 is for Saturday |
| %H | Hour (24-hour clock, e.g. 07) |
| %M | Minutes (e.g. 06) |
| %S | Second (e.g. 05) |

**1.1.1 Timestamps**

**1.1.1.1 Convert timestamps to datetime**

We shall use the strftime() function – see table above. Look at the resources for a more comprehensive list.

**Type the following:**

#Different date and time formats

import datetime as dtime

import time

#Use datetime

ts = dtime.datetime.now()

print(ts.strftime("%Y-%m-%d %H:%M:%S"))

print(ts.strftime("%x %X"))

#ISO format for datetime libraries

print(ts.strftime("%c"))

# Unix timestamp

print("Unix Timestamp 1 ",(time.mktime(ts.timetuple())))

# Get Unix timestamp from a datetime object

datetime\_unix\_time = ts.timestamp()

print('Datetime as unix 2:', datetime\_unix\_time)

# Convert Unix timestamp to datetime object

timestamp = dtime.datetime.fromtimestamp(datetime\_unix\_time)

print('DateTime from Unix timestamp 3: ', timestamp)

#================================================

#Use time

#for greenwich meridian time

ts1 = time.gmtime()

print("\n")

print(time.strftime("%Y-%m-%d %H:%M:%S", ts1))

print(time.strftime("%x %X", ts1))

# Iso Format for time libraries

print(time.strftime("%c", ts1))

**Resources**

<https://timestamp.online/article/how-to-convert-timestamp-to-datetime-in-python>

**1.1.1.2 Current Day, Date and Time Display**

Type the following to obtain current day, date and time dsiplay. This is useful for real time sensor data.

import datetime as dtime

now = dtime.datetime.now()

print("Current day is ", now.strftime("%A"))

print("It is Day number ", now.strftime("%w"), " of the week")

print("Current date is ", now.strftime("%d/%m/%Y"))

print("Another format for current date is ", now.strftime("%d-%B-%Y"))

print("Current time in 24-hour format is ", now.strftime("%H:%M:%S"))

**1.1.2 Time of the day**

**Source:** [**https://english.stackexchange.com/questions/28498/precise-names-for-parts-of-a-day#:~:text=One%20refers%20to%20the%20clock,to%2011%3A59%20pm**](https://english.stackexchange.com/questions/28498/precise-names-for-parts-of-a-day#:~:text=One%20refers%20to%20the%20clock,to%2011%3A59%20pm)

|  |  |
| --- | --- |
| **Time Period (24-hour format)** | **Time of Day** |
| 00:00:01- 11:59:59 | Morning |
| 12:00:00 | Noon |
| 12:00:01-18:00:00 | Afternoon |
| 18:00:01-23:59:59 | Evening |
| 00:00:00 | Midnight |

**Type the following to find out current time of the day based on the table above.**

**Type the following to obtain the time of the day (morning, noon, afternoon)**

import datetime as dtime

now = dtime.datetime.now()

#ctime is current time

ctime = now.strftime("%H:%M:%S")

mNight = "00:00:00"

mtime1 = "00:00:01"

mtime2 = "11:59:59"

ntime = "12:00:00"

aftime1 = "12:00:01"

aftime2 = "18:00:00"

etime1 = "18:00:01"

etime2 = "23:59:59"

#Time of the day

if ctime == mNight:

print("It is midnight now")

else:

print("It is not midnight now")

if ctime >= mtime1 and ctime <=mtime2:

print("It is morning now")

else:

print("It is not morning now")

if ctime == ntime:

print("It is noon now")

else:

print("It is not noon now")

if ctime >= aftime1 and ctime <=aftime2:

print("It is afternoon now")

else:

print("It is not afternoon now")

if ctime >= etime1 and ctime <=etime2:

print("It is evening now")

else:

print("It is not evening now")

* 1. **Find the delta or time duration**
     1. **Find my current age**

Type the following:

#Find your current age

# Figure out your age

# https://gist.github.com/shahri23/1804a3acb7ffb58a1ec8f1eda304af1a

import datetime as dtime

currentDate = dtime.datetime.now()

birthdate = input('Please enter your date of birth (dd/mm/yyyy) ')

bdate= dtime.datetime.strptime(birthdate,'%d/%m/%Y') #strptime - string to datetime object

print (bdate)

daysLeft = currentDate - bdate

print(daysLeft)

#Average number of days per year is 365.242 days

years = ((daysLeft.total\_seconds())/(365.242\*24\*3600))

yearsInt=int(years)

months=(years-yearsInt)\*12

monthsInt=int(months)

days=(months-monthsInt)\*(365.242/12)

daysInt=int(days)

hours = (days-daysInt)\*24

hoursInt=int(hours)

minutes = (hours-hoursInt)\*60

minutesInt=int(minutes)

seconds = (minutes-minutesInt)\*60

secondsInt =int(seconds)

#0:d - the first argument in the tuple and in decimal format, i.e. yearsInt, etc...

print('You are {0:d} years, {1:d} months, {2:d} days, {3:d} hours, {4:d} minutes, {5:d} seconds old.'.format(yearsInt, monthsInt, daysInt, hoursInt, minutesInt, secondsInt))

**Resource:**

[**https://gist.github.com/shahri23/1804a3acb7ffb58a1ec8f1eda304af1a**](https://gist.github.com/shahri23/1804a3acb7ffb58a1ec8f1eda304af1a)

[**https://stackoverflow.com/questions/51491893/how-do-i-get-an-age-in-years-and-date-on-pandas**](https://stackoverflow.com/questions/51491893/how-do-i-get-an-age-in-years-and-date-on-pandas)

* 1. **Time Delta**

A [timedelta](https://docs.python.org/3.3/library/datetime.html#datetime.timedelta) object represents a duration, the difference between two dates or times. The difference between timedelta and relativedelta (next section) is that time delta does not include the arguments for months and years.

*class* datetime.**timedelta**(*days=0*, *seconds=0*, *microseconds=0*, *milliseconds=0*, *minutes=0*, *hours=0*, *weeks=0*)

#Time delta

from datetime import timedelta

import datetime as dtime

#Minus 1 second from current datetime using `timedelta`

ctime = dtime.datetime.now()

ptime1 = ctime - timedelta(hours=0, minutes=0, seconds=1, microseconds=0)

print('Minus 1 second from current datetime ')

print(ctime)

print(ptime1)

print("\n")

#Minus 1 minute from current datetime using `timedelta`

ptime2 = ctime - timedelta(hours=0, minutes=1, seconds=0, microseconds=0)

print('Minus 1 minute from current datetime ')

print(ctime)

print(ptime2)

print("\n")

#Minus 1 hour from current datetime using `timedelta`

ptime3 = ctime - timedelta(hours=1, minutes=0, seconds=0, microseconds=0)

print('Minus 1 hour from current datetime ')

print(ctime)

print(ptime3)

print("\n")

#Minus 1 day from current datetime using `timedelta`

ptime4 = ctime - timedelta(days=1, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 day from current datetime ')

print(ctime)

print(ptime4)

print("\n")

#Minus 22 days from current datetime using `timedelta`

ptime9 = ctime - timedelta(days=22, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 22 days from current datetime ')

print(ctime)

print(ptime9)

print("\n")

#Minus 1 week from current datetime using `timedelta`

ptime5 = ctime - timedelta(weeks=1, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 week from current datetime ')

print(ctime)

print(ptime5)

print("\n")

#Minus 1 month from current datetime using `timedelta`

#Note there is no argument for months

#Minus 1 year from current datetime using `timedelta`

#Note there is no argument for years

**Resources**

[**https://docs.python.org/3.3/library/datetime.html#timedelta-objects**](https://docs.python.org/3.3/library/datetime.html#timedelta-objects)

* 1. **Relative Delta as Time Duration between two time objects**

The timedelta class stores the difference between two datetime objects. The The relativedelta type is designed to be applied to an existing datetime and can replace specific components of that datetime, or represents an interval of time. The order of attributes considered when this relativedelta is added to a datetime is:

* Year
* Month
* Day
* Hours
* Minutes
* Seconds
* Microseconds
  + 1. **Relative Delta**

We shall find the relative delta between two datetime objects. Relative delta can be used to add delta to a specific time.

**Type the following:**

#Relative delta

# https://dateutil.readthedocs.io/en/stable/relativedelta.html

#https://docs.python.org/3.3/library/datetime.html#timedelta-objects

import datetime as dtime

from dateutil.relativedelta import \*

#Add 1 second to current datetime using `relativedelta`

ctime = dtime.datetime.now()

ftime1 = ctime + relativedelta(hours=0, minutes=0, seconds=1, microseconds=0)

print('Add 1 second to current datetime ')

print(ctime)

print(ftime1)

print("\n")

#Add 1 minute to current datetime using `relativedelta`

ftime2 = ctime + relativedelta(hours=0, minutes=1, seconds=0, microseconds=0)

print('Add 1 minute to current datetime ')

print(ctime)

print(ftime2)

print("\n")

#Add 1 hour to current datetime using `relativedelta`

ftime3 = ctime + relativedelta(hours=1, minutes=0, seconds=0, microseconds=0)

print('Add 1 hour to current datetime ')

print(ctime)

print(ftime3)

print("\n")

#Add 1 day to current datetime using `relativedelta`

ftime4 = ctime + relativedelta(days=1, hours=0, minutes=0, seconds=0, microseconds=0)

print('Add 1 day to current datetime ')

print(ctime)

print(ftime4)

print("\n")

#Add 1 week to current datetime using `relativedelta`

ftime5 = ctime + relativedelta(weeks=1, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Add 1 week to current datetime ')

print(ctime)

print(ftime5)

print("\n")

#Add 1 month to current datetime using `relativedelta`

ftime6 = ctime + relativedelta(months=1, weeks=0, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Add 1 month to current datetime ')

print(ctime)

print(ftime6)

print("\n")

#Add 1 year to current datetime using `relativedelta`

ftime7 = ctime + relativedelta(years=1, months=0, weeks=0, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Add 1 year to current datetime ')

print(ctime)

print(ftime7)

print("\n")

**Type the following:**

#Relative delta

# https://dateutil.readthedocs.io/en/stable/relativedelta.html

#https://docs.python.org/3.3/library/datetime.html#timedelta-objects

import datetime as dtime

from dateutil.relativedelta import \*

#Minus 1 second from current datetime using `relativedelta`

ctime = dtime.datetime.now()

ptime1 = ctime - relativedelta(hours=0, minutes=0, seconds=1, microseconds=0)

print('Minus 1 second from current datetime ')

print(ctime)

print(ptime1)

print("\n")

#Minus 1 minute from current datetime using `relativedelta`

ptime2 = ctime - relativedelta(hours=0, minutes=1, seconds=0, microseconds=0)

print('Minus 1 minute from current datetime ')

print(ctime)

print(ptime2)

print("\n")

#Minus 1 hour from current datetime using `relativedelta`

ptime3 = ctime - relativedelta(hours=1, minutes=0, seconds=0, microseconds=0)

print('Minus 1 hour from current datetime ')

print(ctime)

print(ptime3)

print("\n")

#Minus 1 day from current datetime using `relativedelta`

ptime4 = ctime - relativedelta(days=1, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 day from current datetime ')

print(ctime)

print(ptime4)

print("\n")

#Minus 22 days from current datetime using `relativedelta`

ptime9 = ctime - relativedelta(days=22, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 22 days from current datetime ')

print(ctime)

print(ptime9)

print("\n")

#Minus 1 week from current datetime using `relativedelta`

ptime5 = ctime - relativedelta(weeks=1, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 week from current datetime ')

print(ctime)

print(ptime5)

print("\n")

#Minus 1 month from current datetime using `relativedelta`

ptime6 = ctime - relativedelta(months=1, weeks=0, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 month from current datetime ')

print(ctime)

print(ptime6)

print("\n")

#Minus 1 year from current datetime using `relativedelta`

ptime7 = ctime + relativedelta(years=1, months=0, weeks=0, days=0, hours=0, minutes=0, seconds=0, microseconds=0)

print('Minus 1 year from current datetime ')

print(ctime)

print(ptime7)

print("\n")

**Resources**

<https://dateutil.readthedocs.io/en/stable/examples.html>

<https://docs.python.org/3.3/library/datetime.html#timedelta-objects>

* 1. **Time Series data structure**

Panda dates and times will be useful for reading in a dataset with a date column. Convert a date to datetime in pandas.

**Type the following:**

#Dates and times in pandas

import pandas as pd

date = "4th of September, 2025"

dTime = pd.to\_datetime(date)

print(dTime)

print("\n")

**Type the following:**

#Dates and times in pandas

#Extract components of a datetime data structure

import pandas as pd

import datetime as dtime

from dateutil import parser

import calendar

Adate = "4th of September, 2025"

daTime = pd.to\_datetime(date)

print(daTime)

print("\n")

#Split date to its components

date2 = parser.parse(Adate)

print(date2)

print("\n")

#Extract the year, month and day from currenttime object

ctime = dtime.datetime.now()

date3 = ctime.strftime("%Y-%m-%d")

print("Today's date is: ", date3)

#Get Year from Current Date

print("Today's year is: ", ctime.year)

#Get Month from Current Date

print("Today's month is: ", ctime.month)

#Get Day from Current Date

print("Today's day is: ", ctime.day)

print("\n")

#============================================================

#Extract the year, month and day from a chosen time object

my\_string = "2022-05-23 12:23:34"

#Create a date object

my\_date = dtime.datetime.strptime(my\_string, "%Y-%m-%d %H:%M:%S")

print("My chosen date is: ", my\_date)

#Get Year from a chosen Date

print("My chosen year is: ", my\_date.year)

#Get Month from a chosen Date

print("My chosen month is: ", my\_date.month)

#Get Day from a chosen Date

print("My chosen day is: ", my\_date.day)

#Get hour from a chosen Date

print("My chosen hour of the time is: ", my\_date.hour)

#Get minute from a chosen Date

print("My chosen minute of the time is: ", my\_date.minute)

#Get second from a chosen Date

print("My chosen second of the time is: ", my\_date.second)

**Resource:**

<https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html>

<https://www.dataquest.io/blog/python-datetime-tutorial/>

<https://docs.python.org/2/library/datetime.html>

* 1. **Time-based indexing**

Introduction

One of the most powerful and convenient features of pandas time series is **time-based indexing** — using dates and times to intuitively organise and access our data.

Represent the time component in the index of a [**Series**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html#pandas.Series) or [**DataFrame**](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html#pandas.DataFrame) so manipulations can be performed with respect to the time element.

**Type the following:**

#Pandas Time Series Indexed by Time

#Source https://pandas.pydata.org/pandas-docs/stable/user\_guide/timeseries.html

import datetime

import pandas as pd

import numpy as np

#Use pandas to convert all different inputs to datetime format

dti = pd.to\_datetime(['1/1/2018', np.datetime64('2018-01-01'),

datetime.datetime(2018, 1, 1)])

print(dti)

print("\n")

#Generate sequences of fixed-frequency dates and time spans, H is hour

dti\_2 = pd.date\_range('2018-01-01', periods=10, freq='H')

print(dti\_2)

print("\n")

# Resampling or converting a time series to a particular frequency with index

dti\_2 = pd.date\_range('2018-01-01', periods=10, freq='H')

tseries = pd.Series(range(len(dti\_2)), index = dti\_2)

print(tseries)

print("\n")

#Resample the time series to 2 hourly and find the mean

tseries.resample('2H').mean()

**Type the following:**

#Represent time component in the index of a Series or dataframe so that manipulations can be performed with respect

#to the time element

import datetime

import pandas as pd

#D is day

tseries1 = pd.Series(pd.date\_range('2020', freq='D', periods=40))

print(tseries1)

print("\n")

#M is month

tseries2 = pd.Series(pd.period\_range('1/1/2011', freq='M', periods=10))

print(tseries2)

print("\n")

#M is month

#When passed into those constructors.

#DateOffset data however will be stored as object data.

tseries3 = pd.Series([pd.DateOffset(1), pd.DateOffset(2)])

print(tseries3)

print("\n")

tseries4 = pd.Series(pd.date\_range('1/1/2011', freq='M', periods=3))

print(tseries4)

print("\n")

**Missing data**

Pandas represents null date times, time deltas, and time spans as NaT which is useful for representing missing or null date like values

**Type the following:**

#Represent Missing Data for dates, time deltas, and time spans

import datetime

import pandas as pd

missing1 = pd.Timestamp(pd.NaT)

print(missing1)

missing2 = pd.Timedelta(pd.NaT)

print(missing2)

missing3 = pd.Period(pd.NaT)

print(missing3)

#You cannot use equality for NaT

print(missing1 == pd.NaT)

print(missing1 == "NaT")

**Resources**

<https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html>

<https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html>

<https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/>

* 1. **Read in data lists**

Firstly, use pandas Timestamp.

<https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html>

**Type the following:**

#Read in data lists

import datetime

import pandas as pd

#Timestamp for Data List

dates = [pd.Timestamp('2012-05-01'),

pd.Timestamp('2012-05-02'),

pd.Timestamp('2012-05-03')]

print(dates)

print("\n")

#Use datetime and datetime index

dates2 = pd.to\_datetime(['2018-01-05', '7/8/1952', 'Oct 10, 1995'])

dates3 = pd.DatetimeIndex(['2018-01-05', '1952-07-08', '1995-10-10'])

dates4 = pd.to\_datetime(['22/01/10', '18/06/17', '15/12/12'], format='%d/%m/%y')

print(dates2)

print("\n")

print(dates3)

print("\n")

print(dates4)

print("\n")

#Use pandas dataframe

df = pd.DataFrame({ 'year': [2015, 2016],

'month': [2, 3],

'day': [4, 5],

'hour': [2, 3]})

dtime = pd.to\_datetime(df)

print(dtime)

print("\n")

#Pass only the columns we need - Year, month, and day

dtime2 = pd.to\_datetime(df[['year', 'month', 'day']])

print (dtime2)

print("\n")

**Resources**

<https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html>

* 1. **Time Series data visualisation**

Dataset is Air Quality for UCI <https://archive.ics.uci.edu/ml/datasets/Air+Quality>#

**Data Dictionary**

**Attribute Information:**

0 Date (DD/MM/YYYY)  
1 Time (HH.MM.SS)  
2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)  
3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)  
4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)  
5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)  
6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)  
7 True hourly averaged NOx concentration in ppb (reference analyzer)  
8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)  
9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)  
10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)  
11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)  
12 Temperature in Â°C  
13 Relative Humidity (%)  
14 AH Absolute Humidity

**Data Cleansing for Missing Data**

If it is categorical data, replace missing data with mode, while missing numeric data could either be replaced with mean or median. You have to specify what actions you have taken in the assignment report. Note: examples are adapted from

<https://www.tutorialspoint.com/time_series/time_series_tutorial.pdf>

Make sure the AirQualityUCI.csv file is in your DAV folder.

**Type the following:**

#https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/

import pandas as pd

import datetime

#Load dataset

dataset = pd.read\_csv('AirQualityUCI.csv', sep = ";", decimal = ",")

print('Number of records/rows is ', len(dataset))

print(dataset.head())

print("\n")

print(dataset.iloc[ : , 0:14])

print("\n")

#A lot of missing data

#is a null value

print(dataset.isna().sum())

print("\n")

#not a null value

print(dataset.notnull().sum())

print("\n")

**We shall plot temperature against date to view the trend.**

**Type the following:**

#Plot Time Series Graphs

#Covert the date data to datetime object

# We would like to investigate on how temperature changes with respect to time

#Column 0 - date; Column 11 - temperature

#https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/

import pandas as pd

import datetime

import matplotlib.pyplot as plt

Import seaborn as sns

#Load dataset and choose two columns - Date and Temperature, T

dataset = pd.read\_csv('AirQualityUCIvs2.csv', sep = ";", decimal = ",", parse\_dates=True)

dataslice = dataset[['Date','T']]

column1 = pd.to\_datetime(dataset['Date'])

column2 = dataset['T']

print(dataslice)

print("\n")

dataslice2 = dataset.iloc[:, [0,12]]

print(dataslice2)

print("\n")

#Example 1: Plot the time series graph with matplotlib

#This is not a good graph because the date column is not changed to datetime object

dataslice.plot()

plt.show()

#Example 2: Plot the time series graph with matplotlib

#This is the date column has been changed to datetime object

#set the figure size

plt.figure(figsize=(20,10))

plt.scatter(column1, column2, color='green', marker='o')

plt.title("Temperature Change With Respect To Time", fontsize = 20)

plt.xlabel("Date", fontsize = 12)

plt.ylabel("Temperature in Celsius", fontsize = 12)

plt.show()

#Example 3 Plot with Seaborn

plt.figure(figsize=(20,5))

fig = sns.scatterplot(data= dataset, x=column1, y=column2)

fig.set(xlabel='Date', ylabel='Temperature in Degrees Celsius', title = 'Temperature versus Time')

Note: based on the scatter plot, there are some outliers. You could cleanse the data to remove the outliers.

**Resources**

<https://machinelearningmastery.com/time-series-data-visualization-with-python/>

<https://www.tutorialspoint.com/time_series/time_series_tutorial.pdf>

<https://varjo.ktto.fi/files/ekonometrian-jatkokurssi-1498336030292.pdf>

<https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html>

<https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html>

<https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/>

<https://strftime.org/>

<https://www.w3resource.com/python-exercises/python-basic-exercise-3.php>

<https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/>

<https://seaborn.pydata.org/generated/seaborn.scatterplot.html>

1. **Types of Time Series Analysis**
   1. **General Time Series**

<https://towardsdatascience.com/playing-with-time-series-data-in-python-959e2485bff8>

<https://www.dataquest.io/blog/tutorial-time-series-analysis-with-pandas/>

<https://www.earthdatascience.org/courses/use-data-open-source-python/use-time-series-data-in-python/introduction-to-time-series-in-pandas-python/>

**Resources**

<https://www.machinelearningplus.com/time-series/time-series-analysis-python/#:~:text=Time%20Series%20Analysis%20in%20Python%20%E2%80%93%20A%20Comprehensive%20Guide%20with%20Examples&text=Time%20series%20is%20a%20sequence,in%20Python%20%E2%80%93%20A%20Comprehensive%20Guide>.

* 1. **Time Series Forecasting**

Time series forecast could be conducted using:

* Simple time series analysis and forecast
* Multivariate time series forecast
* Bayesian timer series forecast with python
* Classical time series forecast (due to time constraints, we shall only address the highlighted approaches)
  + Autoregression (AR)
  + Vector Autoregression (VAR)
  + Moving Average (MA)
  + Autoregressive Moving Average (ARMA)
  + Autoregressive Integrated Moving Average (ARIMA)
  + Seasonal Autoregressive Integrated Moving-Average (SARIMA)
  + Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
  + Vector Autoregression Moving-Average (VARMA)
  + Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
  + Simple Exponential Smoothing (SES)
  + Holt Winter’s Exponential Smoothing (HWES)
  + Resources <https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/>
* Neural Network
* Regression
* TensorFlow
  + LSTM (Long Short-Term Memory)
  + Deep Learning – Convolution Neural Networks (CNN)
  + Deep Learning - Recurrent Neural Networks (RNN)
  + Resources <https://towardsdatascience.com/3-steps-to-forecast-time-series-lstm-with-tensorflow-keras-ba88c6f05237>
* Prophet – do your own

Timer Series Forecast with Prophet <https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/>

The Prophet library is an open-source library designed for making forecasts for univariate time series datasets. It is easy to use and designed to automatically find a good set of hyperparameters for the model.

**Example 1 Simple time series analysis and forecast**

Save the Superstore.xls file in the DAV folder. Instructions and explanation is embedded in the code. **Types the following:**

#Time Series Forecast

#https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#load dataset and only choose category for furniture

dataset = pd.read\_excel('Superstore.xls')

print(dataset.head())

print("\n")

furniture = dataset[dataset['Category'] == 'Furniture']

print(furniture.head())

print("\n")

#Find the start and end date for furniture order

print('The earliest and latest furniture order dates are: ')

print(furniture['Order Date'].min(), furniture['Order Date'].max())

print("\n")

#Check whether there is any missing data for furniture

print(furniture.isnull().sum())

print("\n")

#Index Time Series Data, Order Date

furniture = furniture.set\_index('Order Date')

print(furniture.index)

print("\n")

#We shall obtain the average sales for each month (MS)

month\_average = furniture['Sales'].resample('MS').mean()

#Only 2017 furniture sales

print(month\_average['2017':])

print("\n")

#Plot the Furniture Sales Time Series Data for all the years

plt.figure(figsize=(20,20))

plt.plot(month\_average, color='green', marker='o')

plt.title("Average Monthly Sales", fontsize = 20)

plt.xlabel("Time", fontsize = 12)

plt.ylabel("Average Sales", fontsize = 12)

plt.show()

#Plot the Furniture Sales Time Series Data for only 2017

plt.figure(figsize=(20,20))

plt.plot(month\_average['2017':], color='red', marker='x')

plt.title("Average Monthly Sales for the Year 2017", fontsize = 20)

plt.xlabel("Time", fontsize = 12)

plt.ylabel("Average Sales", fontsize = 12)

plt.show()

We can also visualise our data using a method called time-series decomposition that allows us to decompose our time series into three distinct components: trend, seasonality, and noise (residual).

Type the following:

#https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b

#Decomposition of time series into components:

#trends, seasonality, noise

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#load dataset and only choose category for furniture

dataset = pd.read\_excel('Superstore.xls')

furniture = dataset[dataset['Category'] == 'Furniture']

#Index Time Series Data, Order Date

furniture = furniture.set\_index('Order Date')

print(furniture.index)

print("\n")

#We shall obtain the average sales for each month (MS)

month\_average = furniture['Sales'].resample('MS').mean()

#Plot the Furniture Sales Time Series Data for all the years

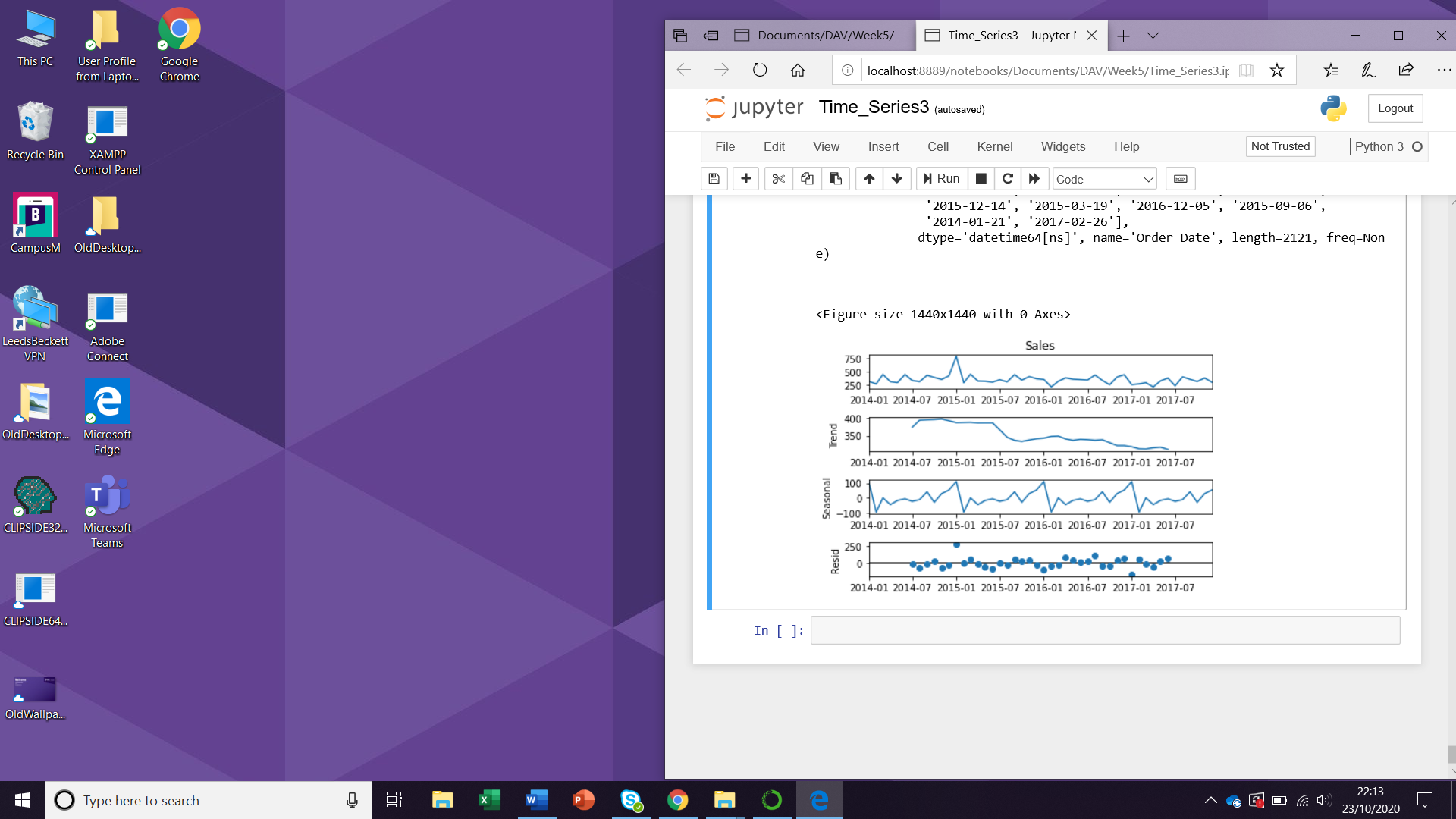
plt.figure(figsize=(20,20))

decomposition = sm.tsa.seasonal\_decompose(month\_average, model='additive')

fig = decomposition.plot()

plt.show()

**Output Display and Interpretation:**



* Downward trend
* Cyclic seasonality
* Residual – rather small

**Resources**

<https://arxiv.org/ftp/arxiv/papers/1302/1302.6613.pdf>

<https://towardsdatascience.com/time-series-analysis-in-python-an-introduction-70d5a5b1d52a>

<https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/>

<https://www.stat.ipb.ac.id/en/uploads/KS/S2%20-%20ADW/3%20Montgomery%20-%20Introduction%20to%20Time%20Series%20Analysis%20and%20Forecasting.pdf>

<https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/>

<https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-python/>

<https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b>

* 1. **Vector Autoregression**

Autoregression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next time step. Vector Autoregression (VAR) is a forecasting algorithm that can be used when two or more time series influence each other (source and theories are found here: <https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/> ). We will look at it further in Regression.

Vector Autoregression (VAR) is a multivariate forecasting algorithm that is used when two or more [time series](https://www.machinelearningplus.com/time-series/time-series-analysis-python/) influence each other. The requirements are: (I) need at least two time series (variables); (ii) the time series should influence each other. Next, we shall evaluate the following: evaluate causation – use Granger’s Causality Test; cointegration Test; test for stationarity.

**2.3.1 VAR Model**

The source is : https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

The dataset that we are going to use is wage\_growth.txt and data dictionary is as follows:

* rgnp **:** **Real** GNP
* pgnp **:** **Potential** real GNP
* ulc **:** **Unit** labor cost
* gdfco **:** **Fixed** weight deflator **for** personal consumption expenditure excluding food **and** energy
* gdf **:** **Fixed** weight GNP deflator
* gdfim **:** **Fixed** weight **import** deflator
* gdfcf **:** **Fixed** weight deflator **for** food **in** personal consumption expenditure
* gdfce **:** **Fixed** weight deflator **for** energy **in** personal consumption expenditure

Ways to visualise time series data before we conduct forecasting

**Visualisation 1: Type the following:**

#https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/import pandas as pd

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#Print off the number of records and columns of the dataset

dataset = pd.read\_csv('flights.txt', sep = ",")

print(dataset.shape)

print(dataset.head())

#Visualise the time series for all the features or other columns

#https://seaborn.pydata.org/examples/timeseries\_facets.html

#Plot each year's time series in its own facet

#Choose color of palette #https://seaborn.pydata.org/tutorial/color\_palettes.html?#highlight=palette

sns.set\_style('dark')

graph = sns.relplot(data=dataset, x="month", y="passengers", col="year", hue="year", kind="line", palette="pastel", linewidth=4, zorder=5, col\_wrap=3, height=4, aspect=3.0, legend=False)

**Output**

The graphs are too small. In order to have bigger graphs, see the next set of codes.

**Visualisation 2: Type the following:**

Type the following:

#https://seaborn.pydata.org/tutorial/axis\_grids.html

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#Print off the number of records and columns of the dataset

dataset = pd.read\_csv('flights.txt', sep = ",")

print(dataset.shape)

print(dataset.head())

#Plot two columns

sns.relplot(

data=dataset, x="month", y="passengers",

col="year",

kind="scatter", col\_wrap=2

)

**Visualisation 3: Type the following:**

Type the following:

#https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/import pandas as pd

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#Load dataset and set index to the date column

dataset = pd.read\_csv('wage\_growth.txt', sep = ",", parse\_dates=['date'], index\_col = 'date')

print(dataset.shape)

print("\n")

print(dataset.head())

print("\n")

#Visualise the data

fig, axes = plt.subplots(nrows=4, ncols=2, dpi=120, figsize=(10,6))

for i, ax in enumerate(axes.flatten()):

data = dataset[dataset.columns[i]]

ax.plot(data, color='red', linewidth=1)

# Decorations

ax.set\_title(dataset.columns[i])

ax.xaxis.set\_ticks\_position('none')

ax.yaxis.set\_ticks\_position('none')

ax.spines["top"].set\_alpha(0)

ax.tick\_params(labelsize=6)

plt.tight\_layout();

**Visualisation 4: Type the following:**

Type the following:

#Seaborn realplot

#https://seaborn.pydata.org/generated/seaborn.relplot.html

import pandas as pd

import numpy as np

import datetime

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

#Load dataset and set index to the date column

dataset = pd.read\_csv('wage\_growth.txt', sep = ",", parse\_dates=["date"])

print(dataset.shape)

print("\n")

print(dataset.head())

print("\n")

print(dataset['date'])

print("\n")

#Convert to datetime

columnX = pd.to\_datetime(dataset["date"])

print(columnX)

print("\n")

#Visualise the data - actual multi dimensional time series for VAR Model

fig, axes = plt.subplots(nrows=3, ncols=3, dpi=120, figsize=(10,6))

fig.tight\_layout(pad = 3.0)

sns.lineplot(data=dataset, x="date", y="rgnp", ax = axes[0,0])

sns.lineplot(data=dataset, x="date", y="pgnp", ax = axes[0,1])

sns.lineplot(data=dataset, x="date", y="ulc", ax = axes[0,2])

sns.lineplot(data=dataset, x="date", y="gdfco",ax = axes[1,0])

sns.lineplot(data=dataset, x="date", y="gdf", ax = axes[1,1])

sns.lineplot(data=dataset, x="date", y="gdfim",ax = axes[1,2])

sns.lineplot(data=dataset, x="date", y="gdfcf",ax = axes[2,0])

sns.lineplot(data=dataset, x="date", y="gdfce",ax = axes[2,1])

sns.lineplot(data=dataset, x="date", y="gdfce",ax = axes[2,2])

**Build a VAR Model for Forecast**

Multivariate time series forecasting is known as **Vector Auto Regression (VAR)**. We can build a VAR based on univariate time series too.

Explanation for the codes can be found here <https://www.statsmodels.org/dev/vector_ar.html>

**Assumption:**  The VAR class assumes that the passed time series is stationary. Trend stationary is a time series that does not exhibit a trend. Seasonal stationary is a time series that does not exhibit seasonality.

**Call the fit method with desired lag order.**

Source <https://www.mathworks.com/help/econ/time-series-regression-ix-lag-order-selection.html;jsessionid=fb80e536b913affbfb5f288526b6>

**Definition of lag order**

The delay between predictor changes and corresponding response changes must be discovered through data analysis. A common modeling approach is to include the history of a predictor at times *t* - 1, *t* - 2, *t* - 3, ..., *t - p*, with the assumption that significant effects on the current response are only produced by recent changes in the predictor. Specification analysis then considers extending or restricting the lag structure, and finally selecting an appropriate *lag order p*.

**Meaning of terms**

diff() - calculates the difference between two elements.

Lag difference is the difference between two consecutive observations (at t-1 and t).

**Example 1: To explore the impact of one variable on another in two time series variables**

#https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/

#https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/

#https://notes.quantecon.org/submission/5db25b54831cf4001af7e506

#https://www.statsmodels.org/dev/vector\_ar.html

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

from statsmodels.tsa.api import VAR

#Load dataset

data = pd.read\_excel('SW2001\_data.xlsx')

#Column: Year and Quarter Number

#Create an index based on date

data.index = pd.DatetimeIndex(data['obs'])

data\_use = data[['Inflation','Unemployment','Fed Funds']]

data\_use.head()

# Plot graphs against time which is the index

fig, axes = plt.subplots(nrows=3, ncols=1, dpi=120, figsize=(6,6))

fig.tight\_layout(pad = 3.0)

#Plot Inflation against Time

data['Inflation'].plot(title = 'Inflation', ax = axes[0])

#Plot Unemployment against Time

data['Unemployment'].plot(title = 'Unemployment', ax = axes[1])

#Plot Federal Funds against Time

data['Fed Funds'].plot(title = 'Federal Funds', ax = axes[2])

#Compute Changes,using log will visit this more in regression. Log will provide information on change.

#diff is the log difference

#dropna - remove missing values

data\_ret = np.log(data\_use).diff().dropna()

# construct model

model = VAR(data\_ret)

#Fit Model using 8 lags

#Lag Order https://www.mathworks.com/help/econ/time-series-regression-ix-lag-order-selection.html;jsessionid=fb80e536b913affbfb5f288526b6

results = model.fit(8)

print(results.summary())

print("\n")

#Plot autocurrelation function

print(results.plot\_acorr())

print("\n")

#Forecast

#https://www.statsmodels.org/dev/vector\_ar.html

lag\_order = results.k\_ar

# forecast 8 periods foreward

print(results.forecast(data\_ret.values[-lag\_order:],8))

print("\n")

results.plot\_forecast(10)

#Impulse Response Function (IRF) - reaction to any change (due to external factor) in the system, orth - orthogonalised

#Impulse responses are of interest in econometric studies:

#they are the estimated responses to a unit impulse in one of the variables.

irf = results.irf(10)

irf.plot(orth=False)

#Cumulative Effect, cumulative response

irf.plot\_cum\_effects(orth=False)

**Example 2: Time series forecast for temperature. Explanation is embedded in the codes.**

**Type the following:**

#https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from pandas.plotting import lag\_plot

from pandas.plotting import autocorrelation\_plot

from pandas import concat

from statsmodels.graphics.tsaplots import plot\_acf

from statsmodels.tsa.ar\_model import AutoReg

from sklearn.metrics import mean\_squared\_error

from math import sqrt

%matplotlib inline

#Load Dataset

dataset = pd.read\_csv('temperature.txt', sep = ",", parse\_dates=['Date'], index\_col = 'Date')

print(dataset.shape)

print(dataset.head())

#Plot dataset

dataset.plot()

plt.show()

#Plot the data on day t and previous day, t-1

lag\_plot(dataset)

plt.show()

#The example below creates a lagged version of the Minimum Daily Temperatures dataset

#and calculates a correlation matrix of each column with other columns, including itself.

values = pd.DataFrame(dataset.values)

dataframe = pd.concat([values.shift(1), values], axis=1)

dataframe.columns = ['t-1', 't']

result = dataframe.corr()

print(result)

#note in the result, there is a strong correlation between the observation and the lag = 1 value

#Plot the correlation coefficient for each lag variable

#This can very quickly give an idea of which lag variables may be good candidates for use in a predictive model

#and how the relationship between the observation and its historic values changes over time.

autocorrelation\_plot(dataset)

plt.show()

#Interpretation the example shows the swing in positive and negative correlation as

#the temperature values change across summer and winter seasons each previous year.

plot\_acf(dataset, lags=31)

plt.show()

#Develop a persistence model

#we want to develop a model to predict the last 7 days of minimum

#temperatures in the dataset given all prior observations.

#We will use the persistence model to compare with the autoregression model

#Test dataset - last 7 days of the dataset

#We would like to make 7 one day forecast

# split into train and test sets

X = dataframe.values

train, test = X[1:len(X)-7], X[len(X)-7:]

train\_X, train\_y = train[:,0], train[:,1]

test\_X, test\_y = test[:,0], test[:,1]

# persistence model

def model\_persistence(x):

return x

# walk-forward validation

predictions = list()

for x in test\_X:

yhat = model\_persistence(x)

predictions.append(yhat)

test\_score = mean\_squared\_error(test\_y, predictions)

print('Test MSE which is Mean Square Error: %.3f' % test\_score)

# plot predictions vs expected

plt.plot(test\_y)

plt.plot(predictions, color='red')

plt.show()

#Autoregression Model

# train autoregression

# split dataset

X = dataset.values

train, test = X[1:len(X)-7], X[len(X)-7:]

model = AutoReg(train, lags=29)

model\_fit = model.fit()

print('Coefficients: %s' % model\_fit.params)

# make predictions

predictions = model\_fit.predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)

for i in range(len(predictions)):

print('predicted=%f, expected=%f' % (predictions[i], test[i]))

rmse = sqrt(mean\_squared\_error(test, predictions))

print('Test RMSE which is Root Means Square Error: %.3f' % rmse)

# plot results

#Predictions from Fixed AR Model

plt.plot(test)

plt.plot(predictions, color='magenta')

plt.show()

#Interpretation - Forecast looks good except for Day 5 with a big variation

**Example 3 – continuation for Example 2.**

**Types the following:**

**#https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/**

**#Create an Autoregressive model**

**# create and evaluate an updated autoregressive model**

**import pandas as pd**

**from matplotlib import pyplot**

**from statsmodels.tsa.ar\_model import AutoReg**

**from sklearn.metrics import mean\_squared\_error**

**from math import sqrt**

**# load dataset**

**dataset = pd.read\_csv('temperature.txt', header=0, index\_col=0, parse\_dates=True, squeeze=True)**

**# split dataset**

**X = dataset.values**

**train, test = X[1:len(X)-7], X[len(X)-7:]**

**# train autoregression**

**window = 29**

**model = AutoReg(train, lags=29)**

**model\_fit = model.fit()**

**coef = model\_fit.params**

**# walk forward over time steps in test**

**history = train[len(train)-window:]**

**history = [history[i] for i in range(len(history))]**

**predictions = list()**

**for t in range(len(test)):**

**length = len(history)**

**lag = [history[i] for i in range(length-window,length)]**

**yhat = coef[0]**

**for d in range(window):**

**yhat += coef[d+1] \* lag[window-d-1]**

**obs = test[t]**

**predictions.append(yhat)**

**history.append(obs)**

**print('predicted=%f, expected=%f' % (yhat, obs))**

**rmse = sqrt(mean\_squared\_error(test, predictions))**

**print('Test RMSE: %.3f' % rmse)**

**# plot**

**pyplot.plot(test)**

**pyplot.plot(predictions, color='magenta')**

**pyplot.show()**

**Resources**

<https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/>

[**https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b**](https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b)

* 1. **Time Series Clustering - this will be visited when we work on clustering**

Time series K-means clustering

Time series hierarchical clustering

Hierarchical clustering and DTW algorithm

**Further Resources**

Data Resampling (aggregate, etc…)

<https://www.w3resource.com/pandas/series/series-resample.php>

**Install Prophet**

<https://anaconda.org/conda-forge/fbprophet>

**Dataset Sources**

<https://archive.ics.uci.edu/ml/datasets.php>

<https://data.world/datasets/cars>